Property-Based Testing

Kinds of Testing

Example-Based Tests

+, -

```
expect(1 + 2).to eq 3
```

```
expect(3 - 2).to eq 1
```

```
expect(3 - 2).to eq 1
```

Property-Based Tests

For some number, n:

$$n + 1 > n$$

For some number, n:

$$n + 1 > n$$

$$n - 1 < n$$

For some number, n:

$$n + 1 > n$$

$$n - 1 < n$$

$$n + 1 - 1 = n$$ #

Practicalities

```
it "describes + and -" do
  property_of {
    integer
  }.check { |n|
    expect(n + 1 > n).to be true
    expect(n - 1 < n).to be true
    expect(n + 1 - 1).to eq n
  }
end
```

```
it "describes + and -" do
  property_of {
    integer
  }.check { |n|
    expect(n + 1 > n).to be true
    expect(n - 1 < n).to be true
    expect(n + 1 - 1).to eq n
  }
end
```

```
it "describes + and -" do
  property_of {
    integer
  }.check { |n|
    expect(n + 1 > n).to be true
    expect(n - 1 < n).to be true
    expect(n + 1 - 1).to eq n
  }
end
```

```ruby
it "describes + and -" do
  property_of {
    integer
  }.check { |n|
    expect(n + 1 > n).to be true
    expect(n - 1 < n).to be true
    expect(n + 1 - 1).to eq n
  }
end
```

```
success: 100 tests
  describes + and -
```

Something that breaks

For some string, x:

```
x.split(" ").join(" ") == x
```

```
it "split/join is reversible" do
  property_of {
    string
  }.check { |x|
    expect(
      x.split(" ").join(" ")
    ).to eq(x)
  }
end
```

```
it "split/join is reversible" do
  property_of {
    string
  }.check { |x|
    expect(
      x.split(" ").join(" ")
    ).to eq(x)
  }
end
```

```ruby
it "split/join is reversible" do
  property_of {
    string
  }.check { |x|
    expect(
      x.split(" ").join(" ")
    ).to eq(x)
  }
end
```

```ruby
it "split/join is reversible" do
  property_of {
    string
  }.check { |x|
    expect(
      x.split(" ").join(" ")
    ).to eq(x)
  }
end
```

```
failure after 7 tests, on:
" &2M1`"
found a reduced failure case:
" &M1`"
found a reduced failure case:
" M1`"
```

```
found a reduced success:
"M1`"
minimal failed data is:
" M1`"
  split/join is reversible (FAILED - 1)
```

```
failure after 7 tests, on:
" &2M1`"
found a reduced failure case:
" &M1`"
found a reduced failure case:
" M1`"
found a reduced success:
"M1`"
minimal failed data is:
" M1`"
  split/join is reversible (FAILED - 1)
```

```
failure after 7 tests, on:
" &2M1`"
found a reduced failure case:
" &M1`"
found a reduced failure case:
" M1`"
```

```
found a reduced success:
"M1`"
minimal failed data is:
" M1`"
  split/join is reversible (FAILED - 1)
```

```
failure after 7 tests, on:
" &2M1`"
found a reduced failure case:
" &M1`"
found a reduced failure case:
" M1`"
found a reduced success:
"M1`"
minimal failed data is:
" M1`"
  split/join is reversible (FAILED - 1)
```

```
failure after 7 tests, on:
" &2M1`"
found a reduced failure case:
" &M1`"
found a reduced failure case:
" M1`"
found a reduced success:
"M1`"
minimal failed data is:
" M1`"
  split/join is reversible (FAILED - 1)
```

Three Things:

Data Generation,
Testing with the Data,
Data Reduction

Uses

Edge Cases

Treating the code like an adversary

Honest TDD;
No fudging code to pass an example test.

Kinds of Properties

Reversible

Reversible

```
n + 1 - 1 == n
```

Reversible

```
n + 1 - 1 == n
```

```
# where y != " "
x.split(y).join(y) == x
```

Reversible

```
n + 1 - 1 == n
```

```
# where y != " "
x.split(y).join(y) == x
```

```
decompress(compress(d)) == d
```

Reversible

```
n + 1 - 1 == n

# where y != " "
  x.split(y).join(y) == x

decompress(compress(d)) == d

 t.to_zone('UTC')
  .to_zone(t.zone) == t.zone
```

Reversible

```
n + 1 - 1 == n

# where y != " "
x.split(y).join(y) == x

decompress(compress(d)) == d

 t.to_zone('UTC')
  .to_zone(t.zone) == t.zone
```

I goofed this in the original; I've fixed this example. Pardon my onstage embarrassment. :-)

Repeatable

Repeatable

```
list.sort.sort == list.sort
```

Repeatable

```
list.sort.sort == list.sort
```

```
handler.handle(evt).handle(evt)
== handler.handle(evt)
```

Unbreakable Rules

Unbreakable Rules

```
list.sort.count == list.count
```

Unbreakable Rules

```
list.sort.count == list.count

list.sort.all? {|x|
  list.find_index(x) != nil
}
```

Swapping the Ordering

Swapping the Ordering

```
a.map{|n| n + 1}.sort
==
a.sort.map{|n| n + 1}
```

Prove a Small Part

Prove a Small Part

```
pairs(list.sort).all?{|(x,y)|
  x <= y
}
```

```
# pairs([1,2,3])
#  => [[1,2], [2,3]]
```

Hard to Solve, Easy to Check

Hard to Solve, Easy to Check

```
solve(solvable_maze)   != nil
solve(unsolvable_maze) == nil
```

Hard to Solve, Easy to Check

```
solve(solvable_maze)   != nil
solve(unsolvable_maze) == nil
```

I really punted on this example, particularly how you generate a maze you *know* is solvable. Sorry.

Consult an Oracle

Consult an Oracle

```
list.hypersort == list.sort
```

Consult an Oracle

```
list.hypersort == list.sort
```

```
new_code(input) == old_code(input)
```

SHOW US SOME REAL EXAMPLES

```
it "can round-trip last-logged-in" do
  property_of {
    (Time.current - float.abs)
  }.check { |time|
    user = User.create(
      username: "Sam",
      last_logged_in_at: time,
    )
    expect(
      User.find(user.id).last_logged_in_at
    ).to eq(time)
  }
end
```

```
it "can round-trip last-logged-in" do
  property_of {
    (Time.current - float.abs)
  }.check { |time|
    user = User.create(
      username: "Sam",
      last_logged_in_at: time,
    )
    expect(
      User.find(user.id).last_logged_in_at
    ).to eq(time)
  }
end
```

```
it "can round-trip last-logged-in" do
  property_of {
    (Time.current - float.abs)
  }.check { |time|
    user = User.create(
      username: "Sam",
      last_logged_in_at: time,
    )
    expect(
      User.find(user.id).last_logged_in_at
    ).to eq(time)
  }
end
```

```
failure: 0 tests, on:
Sat, 13 Jun 2015 04:39:52 UTC +00:00
    can round-trip last-logged-in (FAILED - 1)

Failures:

  1) User round-trip can round-trip last-logged-in
     Failure/Error:
expect(User.find(user.id).last_logged_in_at).to eq
time

         expected: 2015-06-13 04:39:52.835645641 +0000
              got: 2015-06-13 04:39:52.835645000 +0000
```

```
it "after_transition args" do
  property_of {
    array { choose boolean, string, integer}
  }.check { |args|
    test = -> (a) { expect(a).to eq(args) }

    machine = Class.new do
      state_machine initial: :stopped do
        event :go do
          transition :stopped => :going
        end
        after_transition(:stopped => :going,
                         :do => proc { |machine,transition|
                           test.call(transition.args)
                         })
      end
    end

    machine.new.go(*args)
  }
end
```

```
it "after_transition args" do
  property_of {
    array { choose boolean, string, integer}
  }.check { |args|
    test = -> (a) { expect(a).to eq(args) }

    machine = Class.new do
      state_machine initial: :stopped do
        event :go do
          transition :stopped => :going
        end
        after_transition(:stopped => :going,
                         :do => proc { |machine,transition|
                           test.call(transition.args)
                         })
      end
    end

    machine.new.go(*args)
  }
end
```

```
it "after_transition args" do
  property_of {
    array { choose boolean, string, integer}
  }.check { |args|
    test = -> (a) { expect(a).to eq(args) }

    machine = Class.new do
      state_machine initial: :stopped do
        event :go do
          transition :stopped => :going
        end
        after_transition(:stopped => :going,
                         :do => proc { |machine,transition|
                           test.call(transition.args)
                         })
      end
    end

    machine.new.go(*args)
  }
end
```

```
it "after_transition args" do
  property_of {
    array { choose boolean, string, integer}
  }.check { |args|
    test = -> (a) { expect(a).to eq(args) }

    machine = Class.new do
      state_machine initial: :stopped do
        event :go do
          transition :stopped => :going
        end
        after_transition(:stopped => :going,
                         :do => proc { |machine,transition|
                           test.call(transition.args)
                         })
      end
    end

    machine.new.go(*args)
  }
end
```

```
failure: 1 tests, on:
["y}K'ID", "aR/-xm", "^H:/_B", true, false, true]
found a reduced failure case:
["y}K'D", "aR/-xm", "^H:/_B", true, false, true]
found a reduced failure case:
["}K'D", "aR/-xm", "^H:/_B", true, false, true]
...
found a reduced failure case:
["", "", "", true, false, true]
found a reduced failure case:
["", "", "", true, false]
found a reduced failure case:
["", "", "", false]
found a reduced success:
["", "", ""]
minimal failed data is:
["", "", "", false]
```

```
failure: 1 tests, on:
["y}K'ID", "aR/-xm", "^H:/_B", true, false, true]
found a reduced failure case:
["y}K'D", "aR/-xm", "^H:/_B", true, false, true]
found a reduced failure case:
["}K'D", "aR/-xm", "^H:/_B", true, false, true]
...
found a reduced failure case:
["", "", "", true, false, true]
found a reduced failure case:
["", "", "", true, false]
found a reduced failure case:
["", "", "", false]
found a reduced success:
["", "", ""]
minimal failed data is:
["", "", "", false]
```

```
failure: 1 tests, on:
["y}K'ID", "aR/-xm", "^H:/_B", true, false, true]
found a reduced failure case:
["y}K'D", "aR/-xm", "^H:/_B", true, false, true]
found a reduced failure case:
["}K'D", "aR/-xm", "^H:/_B", true, false, true]
...
found a reduced failure case:
["", "", "", true, false, true]
found a reduced failure case:
["", "", "", true, false]
found a reduced failure case:
["", "", "", false]
found a reduced success:
["", "", ""]
minimal failed data is:
["", "", "", false]
```

Random

vs

Exhaustive

Refs, Credits and Things to Look At

- fsharpforfunandprofit.com
  (Property-Based Testing Posts)
- github.com/charleso/property-testing-preso
  (LambdaJam talk)
- Rantly (Ruby, used in examples)
- QuickCheck, SmallCheck (Haskell)
- Hypothesis (Python)

Fin.

Rob Howard
@damncabbage
robhoward.id.au